DOCUMENT RESUME

ED 283 500 IR 012 623

AUTHOR Van Baerle, Susan Lynn
TITLE Computer-Generated, Three-Dimensional Character Animation.
PUB DATE 85
NOTE 78p.; Master's Thesis, Ohio State University.
PUB TYPE Dissertations/Theses - Master Theses (042) -- Viewpoints (120)

EDRS PRICE MF01/PC04 Plus Postage.
DESCRIPTORS *Animation; *Characterization; *Computer Graphics; *Production Techniques; Systems Development; *Three Dimensional Aids

ABSTRACT

This master's thesis begins by discussing the differences between 3-D computer animation of solid three-dimensional, or monolithic, objects, and the animation of characters, i.e., collections of movable parts with soft pliable surfaces. Principles from two-dimensional character animation that can be transferred to three-dimensional character animation are outlined and advantages of working with computer generated animation are discussed. The current means of animating characters at Ohio State University are illustrated by describing the process of creating three separate character animation pieces and the advantages and disadvantages of various techniques are outlined. Topics discussed include shape interpolation; Amorpha and the Fractalites animation; Twixt, a keyframe animation system; the runner animation; the uneven bars animation; Twixt-3, an event driven animation system; the skeleton animation system; the Snoot and Muttly animation; a future low level character animation system; and higher level character animation systems. It is concluded that if a new animation language is to improve the quality of animation, it must enable the animator to: (1) build the original character; (2) specify its hierarchy; (3) define the original motion; (4) easily change the motion; (5) create individualized parameterized models; (6) build the original surface description of a character; (7) manipulate the surface of its individual parts; (8) manipulate the surface as a whole; and (9) view the surface as needed during the animation process. Both a 19-item reference list and a 29-item bibliography are provided. (MES)

**********
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
**********

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
X This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

ED283500

# COMPUTER-GENERATED, THREE-DIMENSIONAL CHARACTER ANIMATION

A Thesis

Presented in Partial Fulfillment of the Requirements
for the Degree Master of Arts

by

Susan Lynn Van Baerle, B.S.

The Ohio State University
1985

Approved by.

Thomas E. Lankard

Adviser
Department of Art Education

Copyright (c) 1985
by Susan L. Van Baerle
All rights reserved.

IR012623

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Susan L. VanBaerle

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."



BEST COPY AVAILABLE

THESIS ABSTRACT

THE OHIO STATE UNIVERSITY

GRADUATE SCHOOL

NAME: Susan Lynn Van Baerle QUARTER/YEAR: Winter 1985

DEPARTMENT: Art Education DEGREE: Masters of Arts

TITLE OF THESIS:
Computer-Generated, Three-Dimensional Character Animation

Computer generated character animation is in its infancy. There are a number of existing systems to animate monolithic objects, however to animate characters there are a number of additions that need to be made to these systems. The objects that are the building blocks of a character need to maintain a relationship to each other and have soft pliable surfaces. The current means of animating characters at Ohio State University, is illustrated by stepping through the process of creating three separate character animation pieces. Advantages and disadvantages of technique are outlined. Potential design of systems for the near and more distant future are proposed.

Thomas E. Linehan

Adviser's Signature

## ACKNOWLEDGMENTS

I would like to acknowledge the support of my adviser Dr. Thomas Linehan and Charles Csuri the director of The Computer Graphics Research Group. I would like to thank all the members of The Computer Graphics Research Group for the lively discussions about computer animation software and their support for the animations I created there. A special thanks goes to the people who wrote software specific to problems I encountered during the production of my animation projects.

John A. Berton transcol (vertex color control)
John C. Donkin blb (bird leg builder)
Julian E. Gomez twixt (animation system)
Douglas L. Kingsbury bird-attach, Pathwalker, bubble flight (bubble motion control)
Edwin Tripp III tog (tubular object generator)

I would like to thank John A. Berton for creating the original music for The Uneven Bars and Snoot and Muttly. I also thank Douglas L. Kingsbury for his effort on the animation of Snoot and Muttly, and Cranston/Csuri Productions for donating calculation time for Snoot and Muttly.

## Table of Contents


Acknowledgements ..... ii
Table of Figures ..... iv
Chapter 1. INTRODUCTION ..... 1
Chapter 2. ANIMATION ..... 3
Chapter 3. SHAPE INTERPOLATION ..... 7
Chapter 4. AMORPHA AND THE FRACTALITES ANIMATION ..... 12
Chapter 5. TWIXT: A KEYFRAME ANIMATION SYSTEM ..... 15
Chapter 6. THE RUNNER ANIMATION ..... 18
Chapter 7. THE UNEVEN BARS ANIMATION ..... 23
Chapter 8. TWIXT-3: AN EVENT DRIVEN ANIMATION SYSTEM ..... 29
Chapter 9. THE SKELETON ANIMATION SYSTEM ..... 33
Chapter 10. SNOOT AND MUTTLY ANIMATION ..... 35
Chapter 11. A FUTURE LOW LEVEL CHARACTER ANIMATION SYSTEM ..... 57
Chapter 12. HIGHER LEVEL CHARACTER ANIMATION SYSTEMS ..... 61
Chapter 13. CONCLUSION ..... 65
List of References ..... 67
BIBLIOGRAPHY ..... 69

## Table of Figures


Figure 1. Example of a three dimensional character ..... 5
Figure 2. Shape interpolation ..... 8
Figure 3. Unexpected shape interpolation ..... 9
Figure 4. Chimera witch transformation ..... 10
Figure 5. Data generation: revolve ..... 11
Figure 6. Examples of final shapes for Amorpha ..... 13
Figure 7. Placement and scale of Amorpha during animation ..... 14
Figure 8. Data generation: tubular object ..... 18
Figure 9. Stick figure drawings for a run cycle ..... 19
Figure 10. Two views of the runner ..... 20
Figure 11. Two raster views of the runner ..... 21
Figure 12. Final data for running cycle ..... 22
Figure 13. Shortening of objects during shape interpolation ..... 23
Figure 14. Shortening of character limbs ..... 24
Figure 15. Inserting data files to fix shortening ..... 25
Figure 16. Gymnast data: original orientation ..... 26
Figure 17. Gymnast data: rotated ..... 27
Figure 18. Frame from The Uneven Bars ..... 28
Figure 19. Attachment of objects in a hierarchical structure ..... 30
Figure 20. Rotations in hierarchical objects ..... 31
Figure 21. Rotations in hierarchical objects ..... 32
Figure 22. Map of environmental elements ..... 36
Figure 23. Top view of environment ..... 37
Figure 24. Original and final character ..... 38
Figure 25. Individual and Assembled pieces for Muttly's head ..... 39
Figure 26. Drawings for Muttly's walk ..... 40
Figure 27. Muttly's parts made with the tubular object generator ..... 41
Figure 28. Data files for Muttly's walk cycle ..... 42
Figure 29. Tail data files for one wag cycle ..... 43
Figure 30. Muttly with all the pieces attached ..... 44
Figure 31. Paths for the birds on the environment map

..... 48
Figure 32. Close up of Snoot's head ..... 51
Figure 33. Overlay of frames from Snoot and Muttly
..... 55

# CHAPTER 1

## INTRODUCTION

Three-dimensional computer animation houses now produce most of their animations for television. It is used in commercials, station identifications, and in introductions to news' shows. The objects used in these animations tend to be solid three-dimensional objects such as the letters "ABC" or "CHANNEL 12". I will refer to these as monolithic objects. Monolithic objects are moved unchanged past the camera. The emphasis of this type of animation is the creation of a dynamically changing relationship between the object and the camera. The flying monolithic objects are usually shot in one continuous camera move. Computer animation is not bound by the physical realities of setting up a camera. The camera can move with total freedom past or through an object. The camera move can start in outer space and zoom into the United States, through a city, into a room and up to an object in one shot. This would be very difficult to do in live action or with other animation techniques.

There are some major differences between animating monolithic objects and animating characters. Characters are a collection of movable parts with soft pliable surfaces. Objects that are a collection of solid parts will be referred to as articulated objects. Single objects with pliable surfaces will be referred to as plastic objects. Objects that are a combination of plastic and articulated objects are composite objects [13]. Most characters are composite objects. The purpose of character animation is to change the relationship of the parts of the characters in such a way that they tell a story or bring a personality to life. These animations have interesting relationships between the camera and the motion. Free space camera moves are used, as well as more traditional cinematic approaches (e.g. pans, dollys, and cuts).

Current systems used to animate monolithic objects need to incorporate new ingredients in order to accomplish character animation. One is the ability to easily control

articulated figures. The second is the ability to manipulate plastic surfaces, These two things, in combination with one another, must be easily controlled at all levels by the animator.

The elements of a good story with unique and lifelike characters will not be changed by the computer. The creativity will come from people. People create the stories, characters and style of the images before they are ever brought to paper or computer. The computer will be used to help realize the creative thoughts, it can't be given total credit for the success or failure of any animation. Even if the computer has artificial intelligence to make some of these decisions, the people using the computer should be able to supercede the computer's decisions. People will be directing the computer to produce what they have in their mind's eye. Animation, whether created traditionally or with a computer, will end up in the same media: film, videotape or laser videodisc. Computer rendered character animation is not a new art form but an old art form under expansion because of developing technology.

## CHAPTER 2

## ANIMATION

There are many principles from two dimensional character animation that can be transferred to three dimensional character animation. These principles are outlined in Disney Animation The Illusion of Life.

The Principles of Animation

1. Squash and Stretch
   movement within a shape exaggerating the elasticity of living flesh
2. Anticipation
   a movement before a major action that allows the audience to anticipate the upcoming action
3. Staging
   the relationship of the characters to each other and the camera
4. Straight Ahead Action and Pose to Pose
   two ways of approaching the order in which drawings will be made
5. Follow Through and Overlapping Action
   parts of a character move at different speeds: movement is continuous from one action to another
6. Slow In and Slow Out
   smooth acceleration and deceleration into and out of movements
7. Arcs
   most motions are not straight but follow some type of curve
8. Secondary Action
   extra action that supports the main action of a sequence
9. Timing
   the number of frames an action takes and the visual outcome

10. Exaggeration
    to make the essential part of a scene or character bigger
11. Solid Drawing
    the animators ability to render the images and make aesthetic decisions about them
12. Appeal
    anything that people enjoy looking at

A more complete description of each of these principles can be found in Disney Animation the Illusion of Life [16].' pp. 47-69.

All of these principles apply to computer animation however the animator does not need to have the skill to render the images. They need to have an aesthetic sense and make decisions about how to change the images. The computers ability to render three dimensional images allows the animators to work with shaded characters in three dimensional environments. The characters and their environments are described mathematically in three-dimensions. This is not an illusion of depth drawn frame by frame, but a synthetic three dimensional space described to the computer with numbers. It is just as easy to have characters move from front to back on the screen as it is for them to move across the screen from side to side. In traditional cel animation, characters often move from side to side, as this is easier to draw and repeat. For example one walk cycle may be created with sixteen drawings. These sixteen drawings are cycled while the position of the drawings or background are moved relative to the camera. A front to back move would necessitate the drawing of every frame because the size of the character constantly changes as it moves away from the camera. In computer graphics animation depth is no longer an illusion but a mathematical reality.

The camera motion is not limited by real space. A character's motion, once set, can be viewed from anywhere. Thus the same walk can go from front to back in screen space or from side to side, depending on where the camera is placed. One programming statement may be used to place the camera in the desired location. The computer doesn't care if you place it below the character, eye level to the character or directly above the character. It is not difficult for the computer to generate an image with

**Figure 1.**
**Example of a three dimensional character**

foreshortening. Perspective views, that were once hard for an animator to draw, are easily rendered by the computer. The camera can be constantly moving. The animator does not have to render perspective drawings of moving characters in a complicated scene with a moving camera. The animator defines the motion of the characters at one stage, the motion of the camera at a second stage and the computer renders the perspective images of the combination.

There are disadvantages to moving a camera in complicated scenes. This would include calculation time, the length of time it takes to calculate one image or frame of animation. This length of time varies from system to system. On the Vax 11/780 utilized by The Computer Graphics Research Group (C.G.R.G.) at Ohio State University, a very simple scene with one character may calculate in shaded color in two minutes. More characters and a scene of greater complexity could take hours to calculate. These are factors that must be considered in the design of characters, backgrounds and camera motion. To save calculation time the camera could remain steady. In such cases the background need only be calculated once. Characters are then shown over the background in much the same way that

traditional animation is created. Unlike traditional cel animation, there is no limit to the number of layers that can be shown on the computer.

The computer also has the ability to calculate images with numerous colored light sources. This allows the animator to play with the dramatic effects of colored light sources to a greater degree than is possible with theatrical lighting. Computers are not limited to real space and the problems of physically placing lights. In the real world lights moving across a stage at ground level rely on the creation of mechanical means of locomotion. In computer graphics the light doesn't physically exist. It only exists as a mathematical model of a light source. The light could be placed on the end of a characters nose and the effect of the light would be seen but not the light itself. Traditional animation does use multiple colored light sources. However the complexity of drawing these effects is a difficult task. This is especially true if the lights, camera and characters are all moving.

Another advantage of computer generated animation is the ease with which changes can be made. The colors, transparency, size, scale, position, orientation, and motion of any object can be changed with relative ease. Changes may occur freely up until the time that final images begin to calculate. Once an animation is correctly set up, the images can be generated by the computer 24 hours a day. There are no inkers or painters. The images are saved on hard disk and later layered over backgrounds or other characters. These composites, are then filmed or recorded onto videotape.

# CHAPTER 3

## SHAPE INTERPOLATION

According to Thomas and Johnston [16] p.47. the first and most important principal of animation is "squash and stretch". This is also one of the most difficult things for an animator to specify to the computer.

Traditional animation is based on capturing the subtle changes within curved lines that define the shape of an object. Motion exists in more than the rotation of the joints. It also exists within the surface that defines the shape. This surface is continuous across the character. The concept of a pliable surface is most easily visualized in facial expressions. When a character smiles, not only do the lips move but also the shape of the cheeks and the eyes change. "Anything composed of living flesh, no matter how bony, will show considerable movement within its shape in progressing through an action." [16] pp. 47-48. "Its not the eyes, but the glance-not the lips, but the smile..." [16] p. 471. In computer graphics there must be a way to control the change of the characters surface. These surfaces should display squash and stretch, as the traditional animators lines depict squash and stretch, if the characters are to feel alive rather than robotic. One way to create this change is to define it in the object's data list.

Shape interpolation is a transformation of one data description to another. The position of point one on object one is transformed to the position of point one on object two over a series of frames. For shape interpolation to work the structure of the two objects must remain constant, thus maintaining a point to point relationship between the two objects. The interpolation between these shapes can be linear over time, or it can contain acceleration and deceleration.

The data sets for shape interpolation could be built by hand. This is not recommended, as any error in the structural description of the data will be noticeable during shape interpolated. A large number of objects might be needed for the type of motion or expressions desired,

**Figure 2.**
**Shape interpolation**

multiplying the potential for human error. If the points of one object are defined in a different order then the points in the second file the data will not change shape as would be expected.

Another way to create the objects is by moving the points that define the first object into the position of the second object. This redefines the shape of the objects without changing the structural description of the data. Shape interpolation can then be run between the original and the new data description. This type of approach was used at the Computer Graphics Lab for the witch in "Chimera".

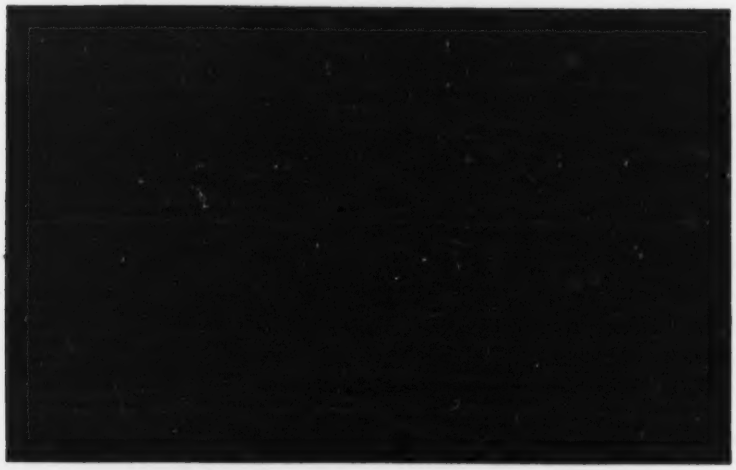

Figure 3.

Unexpected shape interpolation.
The points of the second data file are defined from a new start position on the object.

A complex object might have hundreds of points to locate and move, this becomes a very labor intensive task. Another problem is maintaining a smooth surface on an object after hand moving the points. If a smooth surface is not maintained the lighting model will accentuate the mistake by shading the surfaces associated with the offset point.

A better method is to warp an object, this is done by placing a weight value on the point you are moving. The weight value signifies how the neighboring points will be effected by the displacement of the point you are pulling. In this way moving one point moves its neighboring points in a mathematical relationship to the displaced point. This creates a smooth surface for shading. Another technique used in the CAD industry, is to define an attracter to a set of points. As the attracter is moved, defined points move in a mathematical relationship to it.

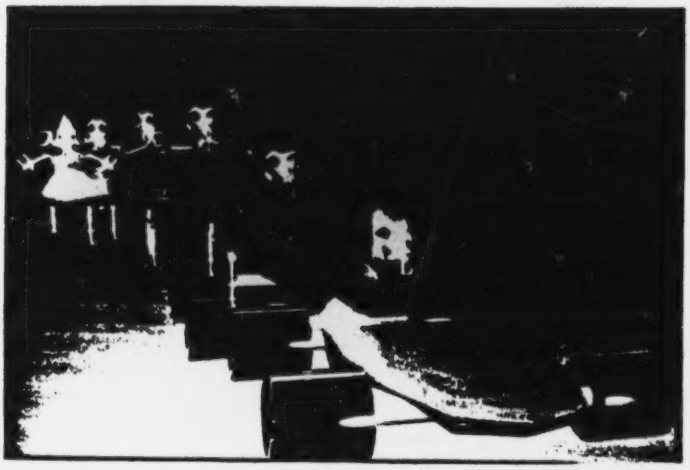

Figure 4.
Chimera witch transformation

The computer can be used to generate the data. This can be done in a number of ways. The simplest method at C.G.R.G. is to build revolved data [3]. To build a revolved object the animator inputs up to four sets of points. These points are then interpolated along a path around the origin. The paths can be of any shape; circular, square or irregular. A polygonal mesh is created from these points. If a consistent number of points are input to each file and a consistent number of positions around the circle is defined, the structure of all the data files created will be identical. This is one of the easiest way to build data for shape interpolation.

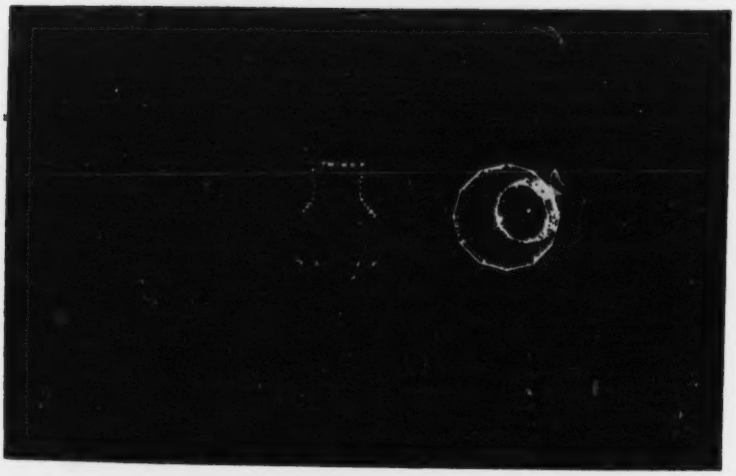

Figure 5.
Data generation: revolve
one input path
two views of the final object

## CHAPTER 4

## AMORPHA AND THE FRACTALITES ANIMATION

Amorpha and the Fractalites is a forty second film of an abstract form that changes shape to music. The music was selected and transferred to sixteen millimeter, magnetic tape and timed before any animation was created. This is the way most traditional animation is done. The picture is matched to pre-recorded sound.

The shapes were designed to match the rhythm and tempo of the music. Once all the shapes were identified on paper, they were to be built by hand. This was done by modeling clay and reading points off an elastic grid stretched over the models. The organic shapes chosen proved to be too complex for this method. The computer data generation program, revolve was used to create the final shapes. This was done by drawing profiles of twelve points each on a bit pad. These profiles were then revolved with twelve sections each. There were desired shapes that could not be generated using the revolve program. It was very difficult to create correctly shaped data with another program which would maintain the identical structure. These desired shapes were modified to fit within the constructs of revolve.

Once the data was built, an animation script was written that would call the shape interpolation program, changeshape. The number of frames between each shape was specified in this script. The timing was already known by counting the frames on the pre-recorded sixteen millimeter magnetic tape.

The first time the animation was viewed, It was decided that the original data files for Amorpha, (the object changing shape) were to conservative. They were then extrapolated to construct a less conservative set of data bases. Extrapolation in this case, refers to shape interpolation that extends beyond the original data file. For example, if changeshape is run from datafile-1 at frame 1 to datafile-2 at frame 10, frame 15 will be a data file that is not between the two original data files. It is one that extends the change in the same linear

**Figure 6.**
**Examples of final shapes for Amorpha**

direction for 5 additional frames. After the data and timing had been set up, Amorpha was manipulated with the basic commands of scale, rotate and place. Amorpha, was placed forward during leaps, rotated during the turns, and scaled to maintain a visually stable volume.

Simple distorted cubes were placed outside of Amorphas path to create an environment around her. The camera positions and center of interest were chosen last, giving consideration to the location of Amorpha, and the composition of the overall scene. This is a different way of approaching animation than traditional cel animation. In traditional cel animation the shot is defined before the animation of the character is begun. Any change in the composition of the shot would necessitate a complete overhaul of the drawings. In computer animation once the motion is established the shots can be changed until final calculation begins. On the system at C.G.R.G. this would be done by changing one line: place eyepoint at x y z. Where x y and z are any location in a three-dimensional co-ordinate space.

Figure 7.
Placement and scale of Amorpha during animation

One problem with this animation was that there was no way to slow in or slow out of the shape interpolation. All the motion was linear and did not have a dance-like quality. This linear motion was manipulated by saving the intermediate interpolated data files. These files were also called into the animation script, but offset from their original timing by a few frames. This created a feeling of slow in and slow out, while created more data files and a more complex script. Additionally it led to small jumps in the animation. These occurred during the switch from one shape interpolations to the next.

The dancer in this piece is a single plastic object. The animation was created by defining ten different objects and interpolating between the shapes of these objects in a sequential order. Rotations and positions were later added to the animation script. The motion produced by the transition from form to form was fluid rather than robotic. This was a first step in learning to manipulate plastic objects.

# CHAPTER 5

## TWIXT: A KEYFRAME ANIMATION SYSTEM

Twixt is an animation language based on the idea of the traditional cel keyframes [7,2]. This animation language allows animators to work visually rather than by programming. The animator positions, rotates and scales an object into a correct keyframe. This keyframe is then "set" and the animator begins to set up the next keyframe. The computer interpolates the parameters of scale, placement and rotation for the object across the designated keyframes. A series of these keyframes are later written out into an animation script. This script is used to produce the final solid shaded images.

The main advantage of this system is that the animator can see the keyframes and preview the motion on a vector display device. The animation can be seen in near real time depending on the display device and/or the complexity of the image. Twixt has two different viewing methods, one with a perspective transform, the other without. The perspective mode gives the animator a truer image, although it takes longer to calculate.

The main disadvantage of keyframe twixt is that every object is set at every keyframe. This system makes overlapping action very difficult. As the animator works setting up the motion of object-one, objects-two, which had not yet been animated, is also being locked into place. The animator must go back into every keyframe and position object two. If the set keyframe was not at the correct frame count, an interpolated position for object-two would have to be calculated and defined at the pre-set keyframe. For example, the desired animation is for a ball to roll down a rocking seesaw. As the keyframes for the rotation of the seesaw are set, the original positions of the ball is being recorded for those keyframes. As the animator begins to set the locations for the ball, it may become obvious that the desired keyframe-numbers for the ball do not match the keyframe-numbers of the seasaw. If the frame numbers do not correspond, the correct location of the ball at the seasaw's keyframes must be computed by

hand and inserted into the existing keyframes. If a camera move for the animation is later added, the position of the eyepoint would have to be hand computed for the existing ball and seesaw keyframes. As the number of objects increases, this system becomes very cumbersome.

This problem was partially solved by adding a command which allowed the animator to interpolate a specific track of an object across all non-controlling keyframes. The computer would calculate the interpolated position for the existing keyframes and insert it into those keyframes for the animator. A track is one parameter of a single object. It could be the scale of the ball in x or the placement in z.

A second early problem was that all the motion was linear. Any change in speed was set by the animator adding keyframes.

The third major problem with this system was the speed at which animation could be viewed. The viewing time for animation in perspective is seldom near real time. The use of simplified data and/or the elimination of the perspective calculation could increase the playback speed to near real time. The accurate timing of an animation is seen by calculating a zoom pan test of the animation. A zoom pan test is a type of pencil test. The frame buffer is divided up into 64 frames. Each of these frames becomes a separate frame of solid shaded animation. The animator runs a program that will zoom into the first of these frames and display it. The program then progresses the display incrementally to the sixty fourth frame. The problem with a zoom pan test is the length of time it takes to calculate. It could take from ten minutes to several hours depending on the complexity. Changes made in the timing must be recalculated and checked again.

Twixt was developed to create motion of monolithic objects, however it needed updating to help solve the problems of character animation. The first update was to allow twixt to access the changeshape program as it calculated the frames. This is done by defining the objects that will be used at specific frames and blending from one file to the next.

Each original data file can be used once, with up to six changes per script.
Following is a sample of shape interpolation in twixt:

```
define object1 as data-1
define object2 as data-2
define object3 as data-3
blend final.objet: data-1 at frame-1, data-2 at frame-10,
data-3 at frame-14
```

## CHAPTER 6

## THE RUNNER ANIMATION

The original intention of this animation was to use plastic surfaces to create realistic human motion. Rather than thinking of human motion as a series of rotations at the joints, it was conceived as a constantly changing surface description. The conventional cel animator animates by changing the shape of a line defining a surface. This animation was done by changing the description of the data defining a surface.

Computer generated data was used for this project. The revolve program did not easily produce shapes similar to that of a human body. There are other types of computer generated data. One of these is a tubular object. Tubular objects are built by making a tube around a

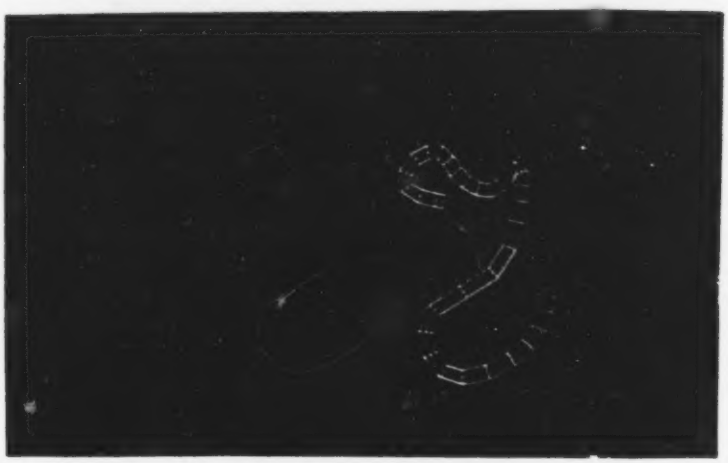

Figure 8.
Data generation: tubular object

connected group of points. The user tells the program what the radius of the tube will be and how many cross sections the tube will have.

To begin this animation, the desired keyframes of a run cycle were drawn on paper with stick figures. Five different tubes were built from the drawings for each runner, two independent legs, two arm that continued down to form the outside of the torso and the head and center section of the torso. These tubes were then combined to create one data file. The left arm and leg were offset negative in z, the torso remained at 0 in z and the right side was offset in the positive direction in z.

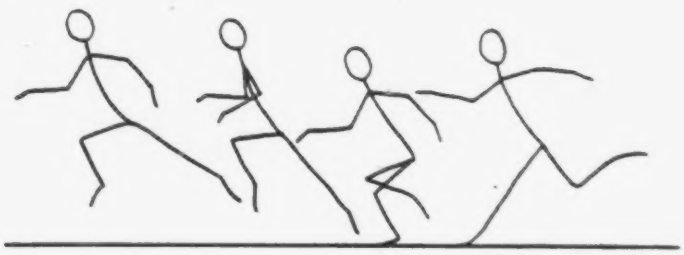

Figure 9.
Stick figure drawings for a run cycle

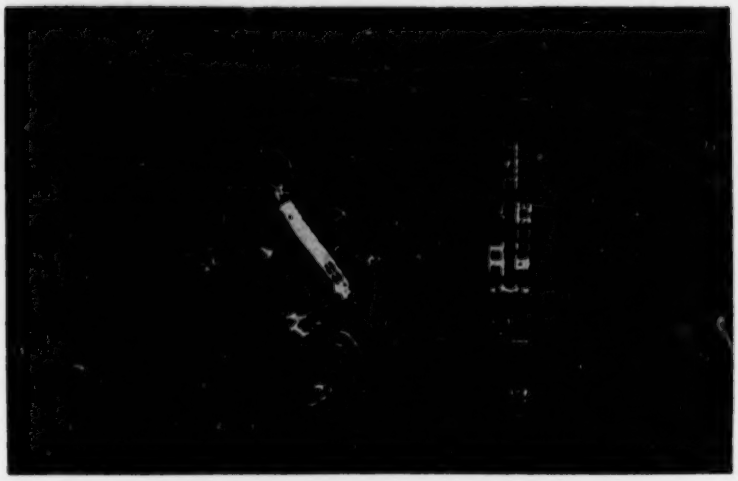

**Figure 10.**
**Two vector views of the runner**
**left: tubes as they were input on the x and y plane**
**Right: tubes placed on the z axis**

The term keydata will be used to describe data files that are in effect the keyframes of an animation. The eight keydata runners maintained a constant data format because the respective pieces were built with the same number of points and the parts of each runner were combined in the same order. By running changeshape between the keydata files, the character did one running step. These data files were than copied and transformed forward the distance of one step. Repeating the pattern created a running cycle.

The animation was defined in twixt and viewed on a vector display device. A thirty frame sequence could be viewed in approximately ten seconds. This system allows the animator to quickly tell if the data files were defined and combined correctly for the changeshape program. Because this display was not real time, it was difficult to tell if the timing of the flight and landing phase on the run were in the proper proportion to each other. The easiest way to view the animation was to run a zoom pan test. At C.G.R.G. this was the only way that an

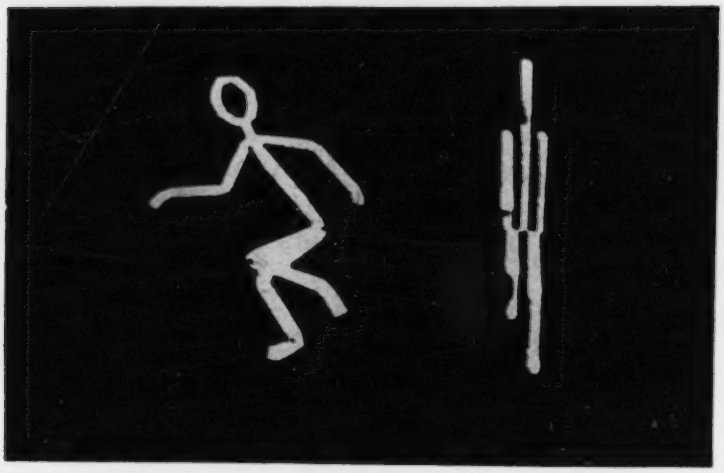

**Figure 11.**
**Two raster views of the runner**

animator could see the timing of an object changing shape in real time. With practice it became easier to get accurate timings on the first or second try.

The most exciting thing about this animation test was that it took only twelve hours to create it. This included making the drawings entering the data, animating and filming it.

Figure 12.
Final data for running cycle

## CHAPTER 7

## THE UNEVEN BARS ANIMATION

A format for displaying believable, complex motion was necessary before manipulation of "the runner" could be expanded. Relying upon my background of coaching gymnastics, an uneven bar routine was designed with several imaginary skills. Stick figures of the routine were drawn on paper, using the same methods as "the runner" animation. The keydata characters were built in an identical fashion. Careful measurements of the length of each part allowed the animator to maintain a constant size throughout the animation. Knowing the standard length of the parts allowed the animator to lengthen or shorten the pieces of the gymnast to suggest stretching or contraction during the gymnastic skills. Maintaining a consistent length in

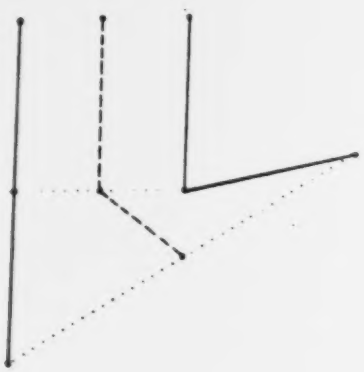

Figure 13.
Shortening of objects during shape interpolation

the characters parts proved to be easier for the animator to control but caused the animation to become somewhat mechanical. The animation lacked a looseness, which made it less expressive.

This method of animating had only one problem while the character was moving in a single direction. Any bend in two adjoining data files could not vary by more than a 45 degrees angle without the pieces becoming obviously shortened. This happens because each point travels the shortest path between the two data sets. This problem was alleviated by inserting more keydata gymnasts between the extremes. A secondary program was written to make the insertion of more keydata files easier. This program would build a tubular object to a predetermined length. For example a new keyframe leg is built by interpolating between two original leg data files. This interpolated point set would be in the correct orientation but shortened in length. The new point file is then run through the program to remake the leg to the correct length. When this leg was inserted in the animation script no

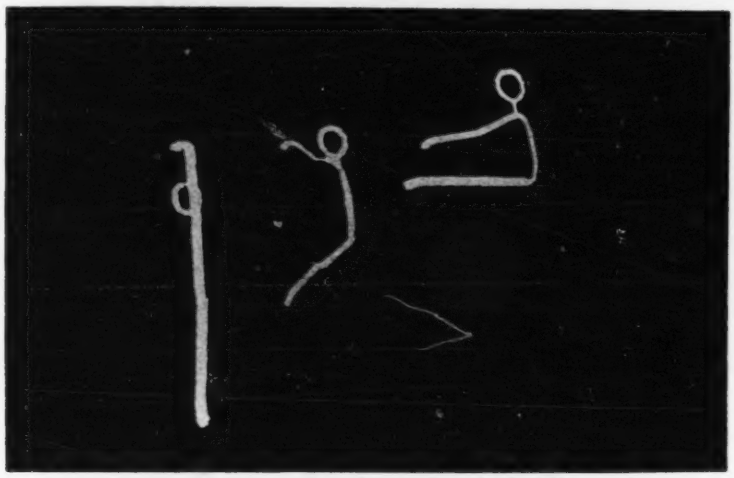

Figure 14.
Shortening of character limbs

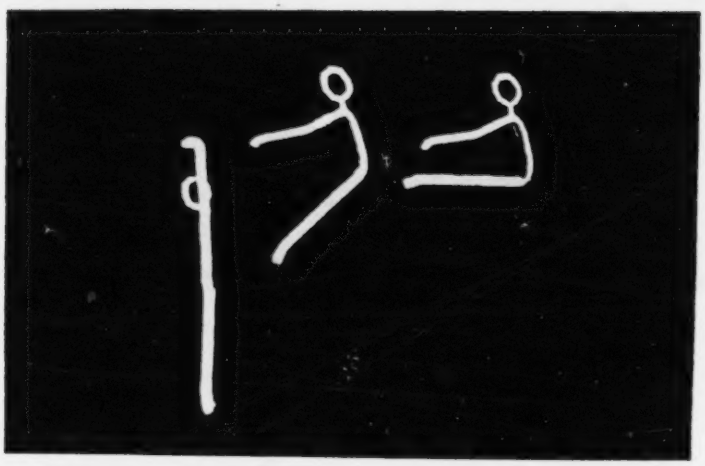

Figure 15.
Inserting data files to fix shortening

shortening would occur. In this way no additional drawings were needed to correct the shortening problem inherent in changeshape.

Another way of avoiding the shortening problem was to built the gymnast in one orientation and then rotated her around the bar. If the rotations around the bar had been built into the data files, the angle change between data files was great. This would necessitate building more in-between keydata files. The data was built without the rotations around the bar, these rotations were added to the animation script in twixt.

The high bar was located at the origin, this made rotations about the high bar simple to implement. Rotations about the low bar were created by rotating the data with the hands at the origin then translating the data to the position of the low bar.

The data files were created facing the correct direction for the animation. This meant that the gymnast did not have to turn to change the direction she was facing. Running change shape on two opposite facing data files creates a transformation of points through the center of the character, but no rotation is needed. The data had to be built facing the original direction of the gymnast and then rotated during the animation. It was difficult to build the data without the correct rotation or direction from where it would eventually be used. To achieve a rotation and still be able to build the data facing the proper direction a gymnast was inserted in the animation. This inserted gymnast was facing the original direction. By rotating that data file to face the opposite direction and substitution an identical data file, facing the new

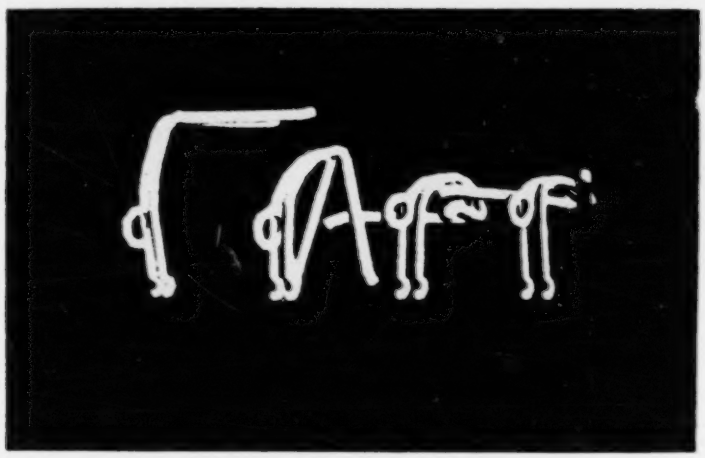

Figure 16.
Gymnast data: original orientation

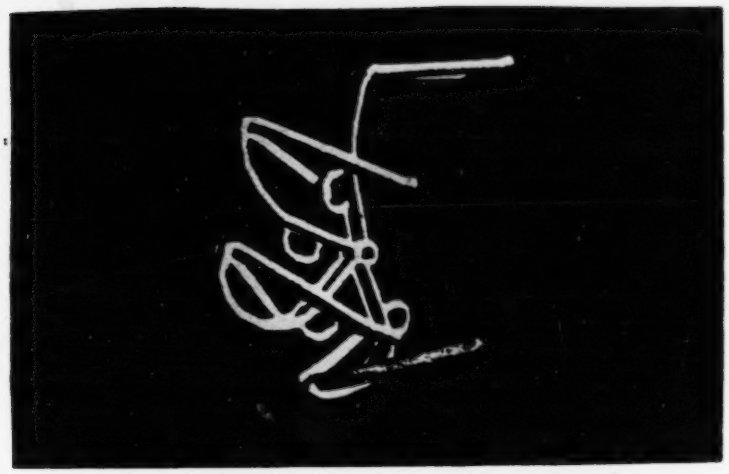

Figure 17.
Gymnast data: rotated

direction, the character would rotate.

The timing on this character was easily changed by changing the number of frames between keydata files. Often changing a sequence by one frame would make a noticeable effect on the motion. For example, as the gymnast swings all the way around the bar the motion was too evenly timed. By removing one frame from the falling portion of the skill and adding it to the rise, a more realistic feeling of weight was created. Timing changes cause one problem, an addition or deletion of one frame number must be continued through every keyframe to the end of an animation script. A large script might have hundreds of keyframe numbers that would have to be moved forward or backward by one number. This was done by hand.

The camera moves were added after the routine was correct. The gymnast had to have her part correct before shots were picked. This was a tedious task as all the eyepoint positions had to be calculated for the gymnasts keyframes.

The uneven bars were the only other objects in this animation, they were also tubular objects. In this

animation, a complex environment would have greatly increased calculation time because the eyepoint was constantly moving.

The data of the gymnast was the key to this animation. There was no scaling and very little placement included. The animation basically consisted of changeshape and rotations. The motion of this animation was fluid rather than robotic. This was not the proper technique for character animation. A great deal of time was spent avoiding limitations in the changeshape program. The next step was to get personalities into characters while making the process less cumbersome. One hundred and nine final data files and many more parts files had been stored on the computer system for this animation. A series of parts used in different combinations could have made the storage space for this project smaller and perhaps less complex. The next project would use a changing combination of parts from a library of objects.

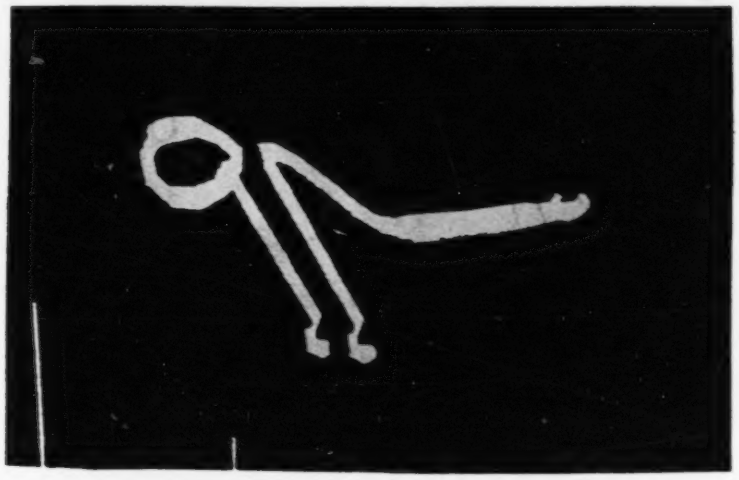

Figure 18.
Frame from The Uneven Bars

## CHAPTER 8

## TWIXT-3: AN EVENT DRIVEN ANIMATION SYSTEM

After the completion of The Uneven Bar animation, work began on an event driven animation system [7,8,5]. The number of calls to changeshape was raised from six per object to two-hundred. This event driven system allows the animator to independently set parameters on objects. It allows for more complex motion descriptions and it contains a means of associating all of a character's parts together.

The animator can specify the position, rotation and scale of any object as an event for that one specific parameter rather than as a keyframe. These events do not effect other parameters for that object or for any other object. The motion of the camera is not affected by the events controlling the motion of the individual parts in the animation. This new feature makes it easier to incorporate both overlapping action and secondary action in animation. Overlapping action is the fourth, and secondary action the eight principals of animation specified in Disney Animation The Illusion of Life [16].

The type of interpolations available in this system were also expanded to include ease in and ease out functions. This ease ability is identical in function to the slow in and slow out principal of animation. Later the ability to specify a spline through any parameter was installed. A spline can be used to smooth out the motion of an object traveling in a curved path. This ability makes it easier to create arced motion on a character while setting fewer events. Arced motion is the seventh principal of animation [16].

Characters may be built out of a group of objects such as a head, arms, legs, torso, and feet. These objects must maintain a relationship to each other for them to remain assembled as independent parts are manipulated. The relationships are maintained using a hierarchical structure. A hierarchical structure refers to the way the manipulation of one object effects another associated object. These hierarchies can be set up in a "tree"

structure. All the branches and leaves of the tree are attached to the trunk. Moving the trunk of a tree effects the placement of all the objects associated with that tree. Moving a leaf on that tree does not effect the placement of the trunk or other leaves. Objects in a hierarchical structure can be animated by rotating, placing and scaling the "limbs". By specifying the order of objects in the hierarchical structure, the animator can maintain a relationship of the character's parts as well as independent control of the desired appendages. For example: if the foot was the trunk of the hierarchy a rotation at the hip of a character, would move the upper part of the torso the head and nose while leaving the legs and feet at the original position. If the parts were not in a hierarchical structure, the nose and head would remain in their original location. They would be unassociated with the torso and would have to be hand placed to maintain their correct relationship. However, flexing and pointing the foot would effect the rotation of the rest of the character, rather than the rotation of the foot.

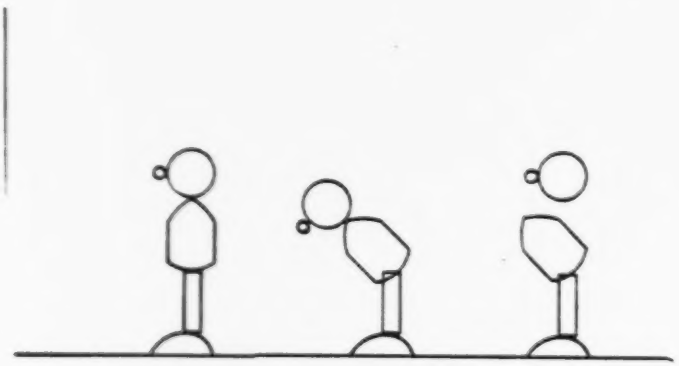

Figure 19.
Attachment of objects in a hierarchical structure

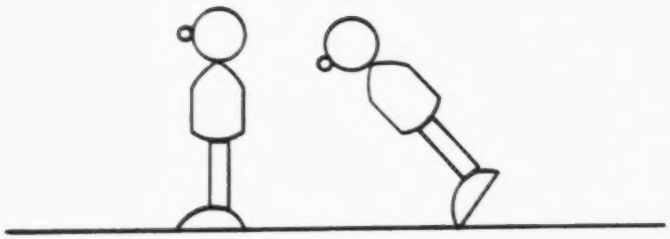

Figure 20.
Rotations in hierarchical objects
the foot as the trunk of a hierarchy

The problem with this association is, that if the torso is the trunk of the tree, rotating the torso forward to begin a walk will also rotate the legs off the ground. The legs would have to be hand rotated back to their original position if the feet were not to slip on the ground as the character walks.

Another problem with the event driven format of twixt is that it output transformation matrixes rather than placement, scale and rotation numbers. This is due to the mathematical complexity of attaching objects. These transformation matrixes however are difficult for the animator to read or hand change.

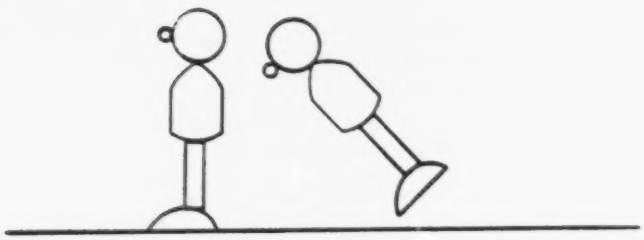

Figure 21.
Rotations in hierarchical objects
the torso as the trunk of a hierarchy

•

Example of a transformation matrix:

```
transform object
          -1.43969   -0.524005   -1.28558   0
          0.160728     -1.3415   0.366804   0
          -1.36915    0.229612    1.43969   0
               0.4           3          8   1
```

# CHAPTER 9

## THE SKELETON ANIMATION SYSTEM

The Skeleton Animation System was designed to allow animators to use a high level language to control the motion of articulated figures [19]. Much of the research for this goal-directed system came from work done in robotics and artificial intelligence. The character in this system is defined with a hierarchical structure. Libraries of motion routines are created and later called, by specifying the goal of a sequence.

The lowest level of this system allows the animator to set up a character's motion by specifying rotations for each joint. For example, in a walk cycle, one command might mean rotate the right knee 70 degrees over 12 frames. Through a series of rotation commands for each joint the character can do one step. Repeating the pattern on the other leg creates one full cycle. The cycle can be repeated as many times as desired. Once set up, the walk cycle is always available by specifying the goal of "walk". At the time of the completion of The Uneven Bars, these rotations were relative not absolute. If the end position of one sequence was not the correct start position of the new sequence, the relative rotations would not create the desired results. The ability to specify absolute rotations was to be later added to this system. This would allow the computer to always begin a motion with the appropriate rotations for the start position.

One of this system's great advantages is the ability to specify a rotation as a support rotation or non support rotation. A support rotation is used when a joint is associated with the object that is the current means of support. For example if the ankle is given a support rotation, the foot stays in one place and the body rotates about the supporting ankle. A non-support rotation of the ankle would cause the body to remain in one position and the foot to rotate.

The system was designed to include artificial intelligence. If the character knows how to walk, the substituting of uneven terrain will not effect it's ability to

move correctly across the surface. This is accomplished, by specifying rotations based on conditional statements. A conditional statement would translate to, when the foot touches the ground rotate the knee.

There were some disadvantages associated with this system at the time The Uneven Bar animation was completed. It only allowed for the manipulation of one character at a time. This makes it difficult to obtain good interaction between characters. It also is not a visually interactive system, this can make it confusing to the non-programmer.

Objects could not be scaled. This meant that there was no way to implement the squash and stretch. Although scaling is not the same as squash and stretch, it can give the illusion of squash and stretch in many situations. The difference between scaling and squash and stretch is the degree of freedom within the manipulation of a surface. For example, when a character runs into a brick wall, the shape of their face may become flattened in varying proportions. The nose may flatten more than the forehead. The face may get wider at the nose and mouth but not across the forehead. This type of surface deformation exhibits true squash and stretch characteristics. Scaling the entire object changes it's size as a unit. In this case the entire head becomes a specified percentage smaller from front to back and a specified amount wider from side to side. There is no elasticity withing the proportions of the single objects. Scaling can have the same overall effect as squash and stretch but it has less detail and expression.

## CHAPTER 10

## SNOOT AND MUTTLY ANIMATION

The original motivation for this film was the creation of personalities in computer generated three dimensional animation. The concept was to choose two very different personalities and have the two personalities play off each other. The two characters in this film are Snoot and Muttly. Snoot is elegant in motion, and motivated by what is proper and looks good. Snoot does not easily give in to frivolous behavior. Muttly is motivated by a playful spirit and doesn't care about looks. The story revolves around Muttly trying to get Snoot to play. Snoot eventually gives into the playful spirit of Muttly, but a slight disguise is needed, before they dance off together.

An environment was built for these two characters to interact. This was built in a manner which would allow quick calculation while still incorporating enough detail to be an entire environment.

There were three types of trees used. These trees were then repeated, rotated, scaled, and placed about the scene to create a tropical environment. One tree was built by combining a series of tapered tubes together at the trunk. Another was created by making a two dimensional leaf shape and warping it into a three dimensional leaf. A series of these leaves were then combined around the trunk to make a tree. The third tree was made by building a tubular object around a leaf shape. A group of these were combined so that they hung from the apex of the trunk. Vertex colors are colors that blend from each separately colored point across the surfaces connecting the points. Random vertex coloring, use a random number generator to assign colors to all of the points. Using a secondary program, these colors were individually weighted toward one of the three primaries red, green or blue. Two Random vertex color files were created for the leaves and the trunks of each of the three basic trees. This duplication of color files allowed for more variety within the forest scene.

The birds' houses were made from 10 different window sections. Each section was comprised of two data files. One was the solid surface of the building and the other a transparent window that fit into the holes of the solid sections. These sections were combined in a variety of ways to create the five houses in the scene. The houses were each colored using random vertex colors. This created a more complex looking environment, without increasing the the complexity of the data. This was done to lessen the the calculation time for each frame. The windows were combined to fit into the houses and painted with a transparency value. This transparency allowed the viewer to see into the houses. The ability to see into the houses meant that the inside walls of the houses had to be created. A surface is only visible from one side. The back side of an object is not visible unless it has been defined. The houses were placed in a semi circle. A map was made detailing the location and size of each tree and house. This was of great help when determining camera locations and the positions of the birds. The environment was 180 degrees of a circle. It was felt that the camera

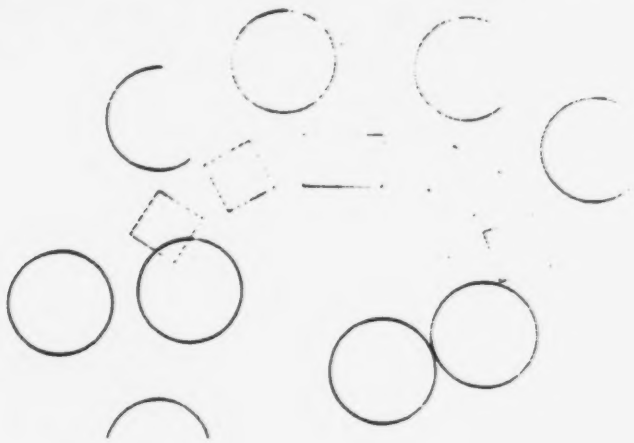

Figure 22.
Map of environmental elements

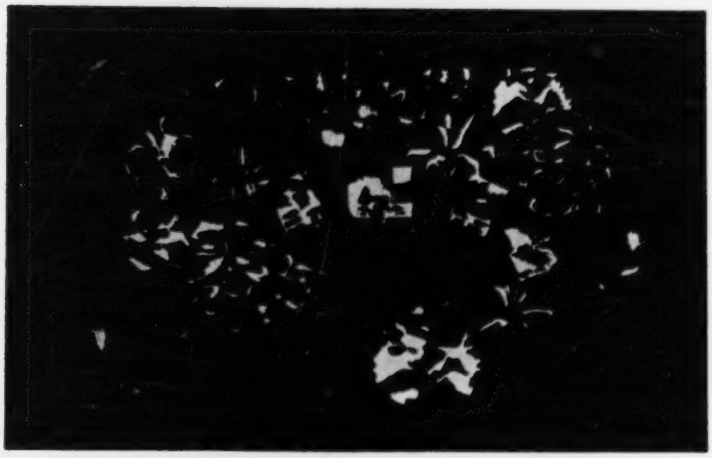

**Figure 23.**
**Top view of environment**

would look at the scene from basically one direction. The environment behind the camera was not originally defined.

Primitive objects were combined to get a basic idea of how the characters would look. These objects were redefined until the final characters evolved.

The head of the birds was made by removing the top polygons of a sphere. The sphere was then rotated to position the hole at the location of the left eye. It was knifed (cut into two separate pieces) on the y axis. The half with the eye was copied and reversed to create the right eye. The two parts were combined to create the head. The head was cleaned with an interactive data manipulation program. This was necessary because the lighting model highlighted the duplicate points located on the center seam of the head. Both birds used the same head.

The eyes were made in three parts: the eye whites, the iris, and the eyelids. The eyelids were made by knifing the head into quarters, retaining the quarter with the eye openings. This piece was scaled smaller than the head and placed inside. The eye whites were done the same way but they used a section of the head that had no openings.

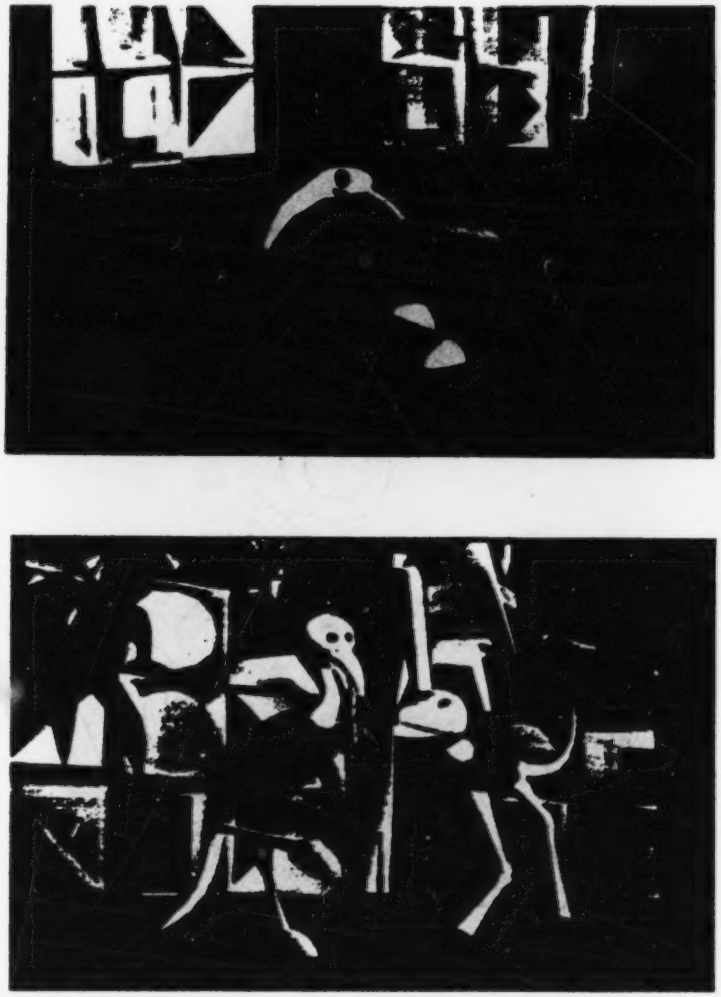

**Figure 24.**
**Original and final character**

The iris was made from a two-dimensional circle wrapped to the radius of the head. Polygon coloring is similar to vertex coloring, but colors for the faces rather than the vertexes of the object are specified. This is how the iris was colored. These pieces were all scaled to correctly fit inside the head.

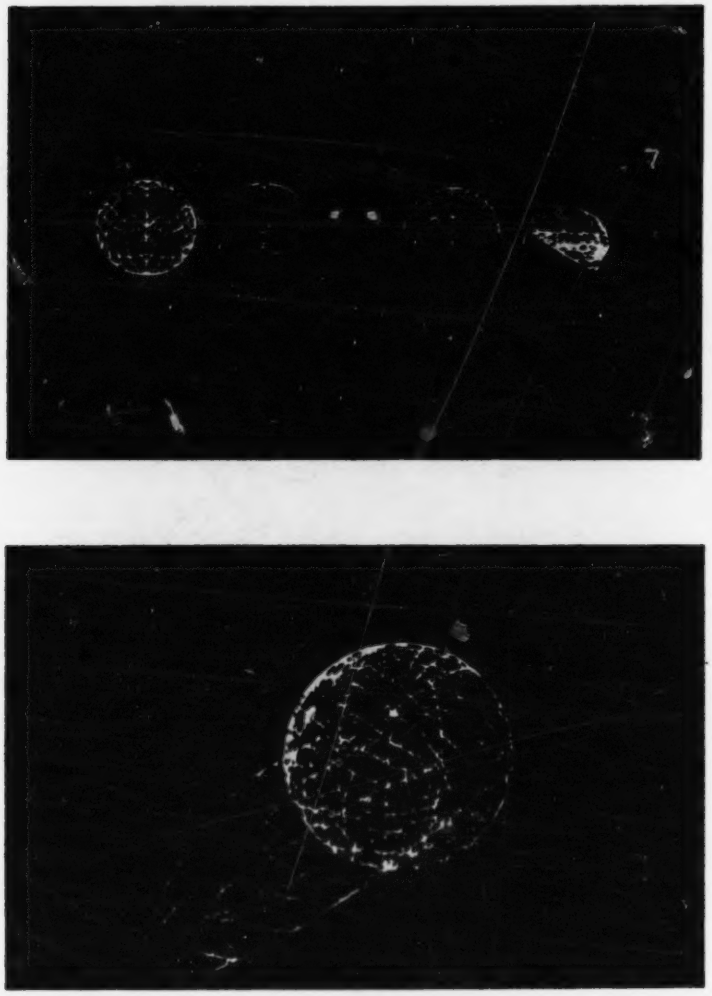

Figure 25.
Individual and Assembled pieces for Muttly's head

The beaks were tubular objects that were knifed to create the top and bottom half of the beak. Different beaks was created for each bird. They were polygon colored so the inside of the mouth was darker than the outside. The bodies were scaled spheres. The feet were created with the revolve program used for the creation of

Amorpha.

The walk cycles were designed in a manner similar to those in the runner and the gymnast. One difference was the direction in which the knee bent. A bird's knee is located much higher than a human's. The joint that appears to be their knee actually correlates to the human ankle, thus it bends the opposite direction of the human knee. The legs could be build anatomically correct. This, however, would have meant animating a hidden joint. The knee which exist up inside the torso. It was decided not to animate a hidden joint.

A new tubular object generator had been written during the production of The Uneven Bars. This program allowed the animator to define the radius for all the points that defined a tube. It also allowed the user to run a spline through these points. In other words, a user could input a few points for an object, run a spline through the object and specify the number of output points. In a separate file, the animator specified the desired radius for the original input points. The tubular

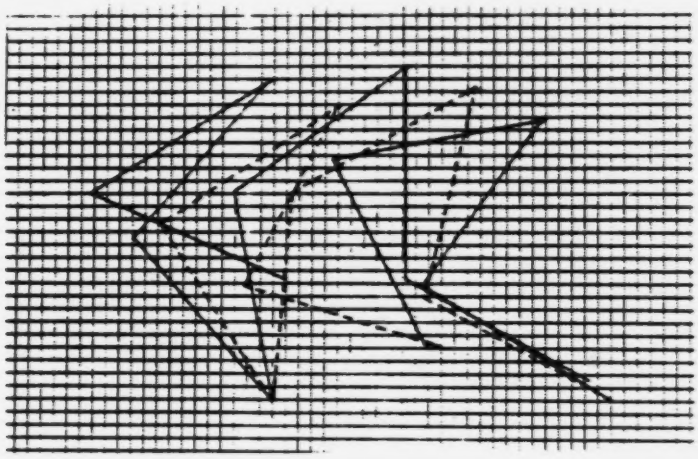

Figure 26.
Drawings for Muttly's walk

object generator then built the tubular object from this information. This allows the animator a greater variety of shapes than the original tubular object generator.

To build the birds' legs, a tube would be built around five known points. A separate radius file listed the radii of those five points: one for the hip, the indent above the knee, the knee, the indent below the knee, and the ankle. Later a special purpose program was written that allowed the animator to specify only three input points: the hip, the knee and the ankle. This program calculated the two points for the indent of the knee, by knowing the distance from the knee to these points. The two birds had separate radius files. This allowed the legs to be made in the same manner but with more characteristic results for each bird. Each bird had it's own walk cycle. These cycles were checked carefully before being used. Any error in one step would be displayed repeatedly in a cycle of that step. This repetition makes it easier to notice errors.

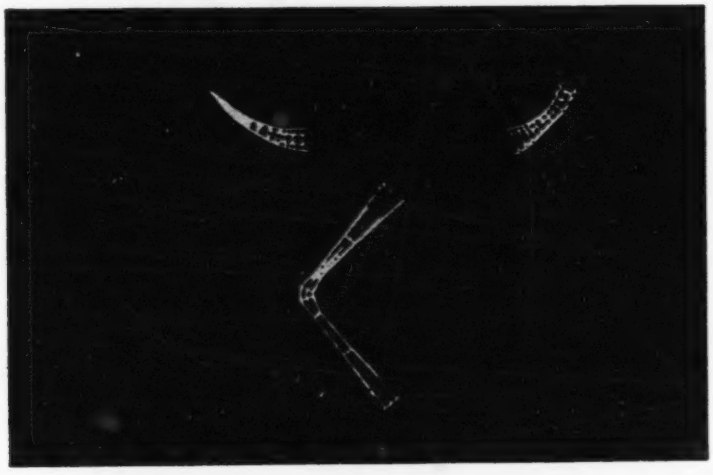

Figure 27.
Muttly's parts made with the tubular object generator

**Figure 28.**
**Data files for Muttly's walk cycle**

The necks and tails for the birds were also made with the tubular object generator. The tail was built by defining six points and running a spline through them. The radii were then diminished out to the tip of the tail. Shape interpolation was run on a series of from six to ten tails which would create a wag. The changeshape program worked well on the data files for the tails. However, because of the motion needed for the necks, some new problems became apparent.

All of the original input point for the legs were described in two-dimensions. The tail moved in three dimensions but because of the complexity of the motion a number of keydata files were always used. The problem with the neck changing shape was caused when the input points moved into the third-dimension. This was done so that the neck would turn to the sides, rather than move just up and down. The linear point to point transformation of the changeshape program did not give good results when the original input points were in three-dimensional space. The problem was similar to the shortening problem described earlier. The shortening problem was due to a change in angle of more than 45 degrees in two-dimensional

**Figure 29.**
**Tail data files for one wag cycle**

space. If the original paths were built in three-dimensions, the limiting angle became significantly less. The data transformed point to point which caused the necks to become very narrow as the birds turned to look to the sides. The solution to this problem was to continue to add key necks until the interpolation produced the desired results. In some sequences there was a different neck data-file for every frame of animation.

Once all the data was built and tested, the parts of the bird were attached to each other in a hierarchical order in the animation script. The order was: the left leg to the right leg, the body to the right leg, the right foot to the right leg, the left foot to the left leg, the neck to the body, the head to the neck, the eye parts to the head, and the tail to the body. This meant that by placing the right leg forward the entire character would move as a unit.

The legs, tail and neck were changing shape as the eyelids, iris, feet and head were manipulated in the animation system twixt. This manipulation of the articulated parts often took a great deal of time. Unlike the

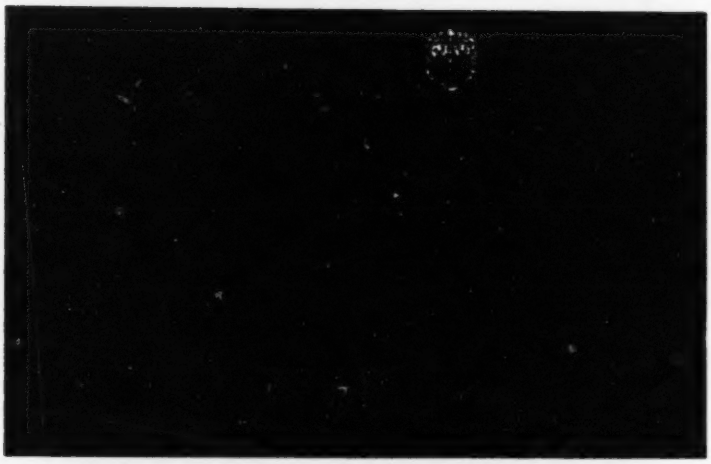

**Figure 30.**
**Muttly with all the pieces attached only points displayed**

changeshape cycles which could be edited on paper, most of these rotations were edited after being viewed on a vector display devise. An animator would often spend a great deal of time waiting for the display device. This problem is magnified if the animator is waiting for an entire animation sequence.

The hips on these characters needed to rotate to give the feel of shifting weight as they walked. Rotating the torso to achieve this look, also rotated all the objects that were attached above the torso. This made the head and neck move back and forth. The desired results was a swaying of the hips, not the head. This problem was solved by using changeshape on the torso. Two more torsos were made for each bird. One was rotated for the left step and the other rotated for the right step. By using changeshape between these three files, the birds' hips rotated without affecting the head or tail of the character. The animation system twixt does not "know" that the data file for the torso has changed. Twixt only processes changes in scale, rotation or position that are defined within an animation script..

This inability to receive information about the data files on a frame by frame basis caused problems with the leg animation. At the first frame in an animation sequence the parts of the bird would all be correctly attached. However, as the data files changed and the legs moved forward, they walked out from under the rest of the character. This problem was solved by feeding the animation script into another program called reattach. This program calculated the distance that the hip of the leg had transformed from the first to the second data file. It then re-attached the body to the leg, offset by the amount of that change. This program reattached the feet to the bottom of the legs, offset by the amount of change for the point at the ankle.

The neck also had this problem because the head was attached to it. The same program was used to re-attach the head to the endpoint of the neck. The tail had nothing attached to it, so it was not effected by this problem.

Once the system was designed to animate the characters, a complete storyboard was made. This storyboard specified action rather than composition. The exact location of the characters would not be known until the stride length and frame count per step was known. Because of this is was difficult to draw the storyboard with correct framing. However the storyboard did specify where there would be close up, medium and long shots.

Six more leg cycles for Muttly were eventually created: 1. running, 2. jogging, 3. skipping, 4. chasse, 5. kick step, 6. Muttly mimicing Snoot's walk. There were five cycles for Snoot: 1. walk, 2. snooty walk, 3. skip, 4. jogging, 5. running.

When the leg cycles and story board were completed. Timing sheets were made with specific frame numbers. This timing sheet made it easy to coordinate all the objects changing shape and the placement of the legs.

Example of a Timing Sheet:

Frame = frame count
R-leg = name of right leg used at current frame count
RL-pos = placement along x axis of the right leg
L-leg = name of left leg used at current frame count
Neck = name of neck used at current frame count
Tail = name of tail used at current frame count

| Frame | R-Leg | RL-pos | L-leg | Neck | Tail | Body |
|---|---|---|---|---|---|---|
| 200 | l_47 | .0 | l_47 | n_41 | t_06 | b_1 |
| 230 | l_47 | .0 | l_47 | | | |
| 238 | l_21 | .1 | l_26 | n_41 | t_07 | b_2 |
| 243 | l_22 | .2 | l_27 | n_14 | t_08 | |
| 248 | l_23 | .3 | l_28 | | t_09 | b_1 |
| 253 | l_24 | .4 | l_29 | | t_10 | |
| 258 | l_25 | .5 | l_30 | n_06 | t_11 | |
| 263 | l_20 | .63 | l_19 | | t_12 | b_3 |
| 268 | l_08 | .82 | l_02 | | t_01 | |
| 270 | | | | | | b_1 |
| 272 | l_09 | .97 | l_03 | n_14 | t_02 | |
| 275 | l_10 | 1.16 | l_04 | | t_03 | b_2 |
| 280 | l_11 | 1.36 | l_05 | | t_04 | |
| 282 | | | | | | b_1 |
| 285 | l_12 | 1.62 | l_06 | n_06 | t_05 | |
| 290 | l_01 | 1.73 | l_07 | | t_06 | b_3 |
| 294 | l_02 | 1.92 | l_08 | | t_07 | |
| 296 | | | | | | b_1 |
| 298 | l_03 | 2.07 | l_09 | n_14 | t_08 | |

This animation pipeline was working, until the character had to walk in some direction, other than along the axis in which the leg cycle had been created. An attempt was made to have the characters turn during a walk cycle. This caused the birds feet to slide across the ground as if on ice skates.

A special purpose program was written to solve this problem, it was called pathwalker. The input to pathwalker was an animation script, which contained a leg cycle, and a path. The paths were originally outlined on

the detailed map of the environment. In this way the characters would not run into each other or environmental elements. The pathwalker program would place and orient the torso along the input path. The torso would now dictate the location of all the other pieces of the character. The hierarchical order of the character was then changed. The legs were attached to the torso rather than the torso to the right leg. The location of the strike point for the leg was recorded and all the key legs for that hit sequence of the step were rebuilt retaining that original strike point on the ground. This was done by transforming the leg back to zero on x and z, then locating the transformed position of the desired hit location. The following legs of that hit sequence were then rebuilt with the foot maintaining the strike locating for the specific path. These new legs were moved back to their desired location. This created many new legs. A new leg was created to replace the original for each of the keylegs that touched the ground throughout the length of every path.

The pathwalker program determined that all the legs would be built with more consistency. The consistency needed was for every leg to be originally built with the hip at zero on the x and z axis. The legs that had already been defined were transformed to meet this criteria. They were placed the proper distance out along the x axis during the animation. It was much easier to cycle these legs if they were placed a consistent distance forward in both timing and position through out the cycle.

Cycling the same set of legs at new positions, created cycles of any length. The length of the path would have to correspond to the length of the particular cycle that was being fed into pathwalker. This was done by measuring the length of the path before it was used in the program.

The characters now needed to begin to interact with each other. The animator would know the length of each of the different birds steps and the frame count associated with them. They would calculate what path would be needed for each bird. The two paths would correlate so that at a specific frame the birds had a particular physical relationship to each other. One sequence might have Muttly doing one walk step, one jog step, one run step and finish within reach of Snoot's beak. During the same time Snoot

would do five walking steps. With a flexible ruler, paths of the correct length were drawn until the end position were in the proper relationship to each other.

One of the early problems with the pathwalker program involved animating the character backing up. The orientation of the torso was always forward along a path, when a character was to step backward, the program would turn the character around 180 degrees and have it step forward. The ability to move backwards was eventually programmed into pathwalker. Another problem associated with this method of orienting the birds was locating the beginning and ending of animation scripts. The orientation of the torso would have to be maintained throughout a cut or sequence. This caused problems if one sequence was in two different scripts. The final orientation of the character on one path might not match with the original orientation of the character on the next path. Separate scripts were created for each sequence of action. When this became too lengthy a script, a new script could be started only at a cutting point. This caused some of the scripts to be

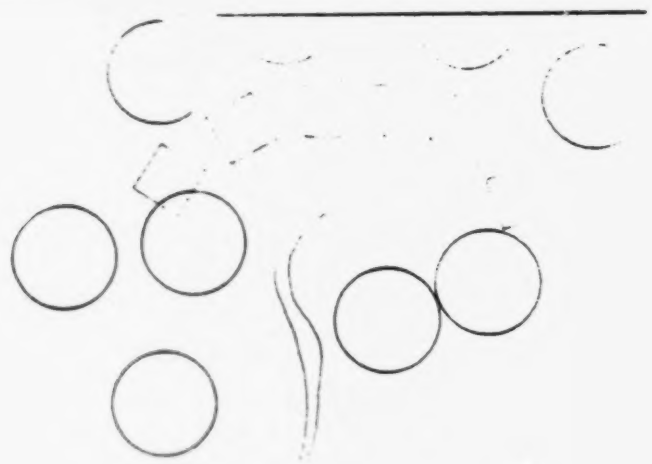

Figure 31.
Paths for the birds on the environment map

extremely large, often between 30,000 and 60,000 lines.

When we began this animation twixt was still a keyframe animation system. All the supporting software, such as the reattach program, were built based on the output from the then-current keyframe system. When the new event driven version of twixt was released, we were tied to animating the changeshape portion of the birds in the keyframe system. This would allow the scripts to be run through the auxiliary programs reattach and pathwalker. The output script were then transformed to the new event driven format of twixt The event driven version of twixt made it easier for the animator to set up articulated motion with overlapping action, thus giving more natural movement.

This process worked well going from keyframe twixt to event driven twixt. However event driven animation scripts could not be returned to keyframe twixt format. The animators had to be certain that their script was correct before changing it to the event driven format of twixt. Any changes made within an event driven animation script could not be maintained if the animator needed to go back to the keyframe version. This became very cumbersome, but due to existing deadlines the software was not changed.

Once an animation script was in the event driven format of twixt it was ready for the final touches. The script contained one bird with the legs moving on the correct path. The feet were attached in the proper places, and there was no slippage on the ground. The body was rotating and the neck was turning. The head was attached but not rotating. This script was then read into a vector display devise. The animator would add the rotations for the birds feet, the head, and eyes. The legs and necks, could not be changed at this point. Changing these data file would cause the associated objects to be incorrectly attached.

The head was built facing the default forward, positive on the z axis. The rest of the bird was built facing toward the right, forward in the positive direction on the x axis. This meant that the head of the bird was rotated 90 degrees about y to face forward. This was done to most easily picture in what direction to rotate the iris of the eyes. Both eyes used the same object, the right eye was rotated in the positive direction about y, the left in a negative direction. They were rotated an equal amount

about x for movement up and down.

The rotation of the head to the forward position created a problem when the head was to tilt as a curious dogs does (on the z axis). This problem resulted because, no history of rotations was maintained. Rotating the head 90 degrees about the Y axis brought the Z axis into the same alignment as the x axis. A rotation around the desired axis Z, would appear the same as a rotation about the x axis. For the first half of the animation the solution to this problem consisted of rotating the head some amount less than or greater than 90, before rotating around the z axis. For the second half of the animation object oriented rotations were implemented. With object oriented rotations, a history of rotations is maintained. The object's coordinate system moves with the object.

When the animator had the script ready, the second bird's script would be read in with the first. At this point the eyes rotations would be finalized. On the vector display devise it was often difficult to tell if the birds were really looking directly at each other. Test frames had to be calculated in raster to make certain that the eyes were lined up correctly.

Another problem with the eyes was that the difference in the scale of all the pieces had to be greater than desired. This was to avoid intersection problems between the layers caused by the display algorithm. The scale at which the pieces began to intersect changed depending on how large the birds heads were on the screen. This meant that the scale for the eyes parts were animated as the characters changed screen size during an animation. The viewer was also able to see into the birds' heads when the eyes were viewed from certain angles. Final scales for the eyes were done after the camera position for each sequence was determined.

The bubbles were animated with a special purpose program. This program randomly placed the bubbles at a vertex within a grid structure. By defining how often the bubbles could move from vertex to vertex, the amount of motion within the group of bubbles could be controlled. This grid was then moved along a path. The path was drawn on the detailed map, which also contained the path for each of the birds. If the bubbles were surrounding the birds, the grid was divided and placed on the sides and above the birds. This alleviated intersections between the birds and

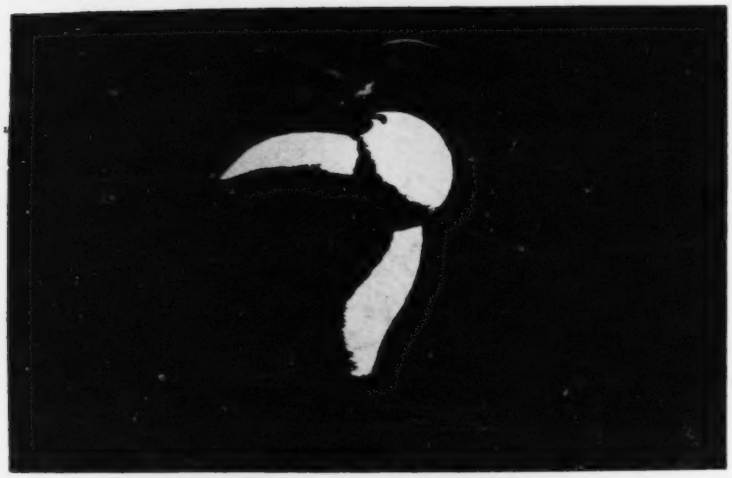

**Figure 32.**
**Close up of Snoot's head**

the bubbles.

The cinematic editing of this animation was done in a sequence by sequence order. The basic eyepoints were known before the animation was begun. Exact eyepoints were picked once the animation of the characters for each section was completed. Consideration was given to the motion of the characters across the screen space and the location of the camera in proceeding and following shots. In this way screen direction was maintained throughout the piece, while obtaining a variety of camera shots. In some shots the camera was placed near the houses looking out of the environment. This part of the environment did not exist and had to be created. This was done to show the best angle on the characters. This camera angle put the characters entirely into shaded areas of the scene. In order to retain consistent lighting throughout the piece the light was not moved. This caused the characters to appear two dimensional as there was very little shading across the characters. The ambient light (the amount of light that is in an environment but not coming directly from the light source) was manipulated to exaggerate existing shading across the characters.

There were usually five layers to each sequence in the animation. The environment had more points and polygons than the display algorithm could handle. Because of this fact it was split into two parts, the ground and half the trees were one part. The houses and the other half of the trees made up the second. The birds were calculated separately. The bubbles were calculated as one unit unless they surrounded the birds. The pieces were all calculated and saved. The ground and background trees were shown first. The houses and foreground trees shown over that. Next the characters were added. The order of this would change depending on how the characters and the bubbles overlapped. The final composite image was recorded onto an electronic still storer. When an entire sequence was recorded on this machine, it was transferred to one inch video tape.

The sound track was created by John A. Berton on a Yamaha DX7, after the animation was completed. Individual sounds were picked for the characters, these sounds were mixed together to accent the characters motions while still retaining a musical quality.

Perhaps the most unexpected part of this animation was the amount of bookkeeping that had to be done. Some routine issues needed to be addressed for each sequence:

(1) In what directory are all the necessary legs?

(2) Which necks go with which legs on each cycle?

(3) What rotated body go with each legs?

(4) What tails can be used that won't intersect the legs? (the tail wags right when the left leg is back)

(5) Where is the other bird at each frame?

(6) Are the frame numbers for all the layers consistent?

(7) What neck will create the correct motion as the bird turns his head? How far will the neck rotate to look in the proper direction?

(8) Will these necks change shape correctly? if not which do I insert?

(9) Are the iris rotations synchronized?

(10) Is the scale on the head,iris,eyelids, and eye whites correct for the camera distance?

(11) Do the bubbles intersect each other or the birds?

(12) Do any characters need to be calculated together because of the way they overlap?

(13) Is the screen direction correct from shot to shot?

(14) Are the characters totally in the shade and looking two dimensional?

(15) Did I edit all the files to make updates for the differences between twixt and the display program?

(16) Is there space on the system to save the calculated frames?

(17) Are all the parts calculated correctly?

(18) Are they on the system? Will they all fit on the system?

(19) Is there anywhere that I need to change the order in which I am showing the layers?

This is by no means a completed list but it will give the reader some idea of how complex animation can become. The computer is very useful in helping to keep order within all this information. Some of our problems were created by our not sticking to the bookkeeping system we had designed for ourselves. Approximately 20 percent of the animators time was spent actively animating this film, the other 80 percent of the time was divided between many other tasks. These included: writing software specific to this animation; avoiding existing software problems; bookkeeping; waiting for access to equipment and waiting for devices to display images. The animators also had the responsibility of checkers, cleanup, and the camera department.

The basic flow chart for this animation was as follows:

(1) Draw the legs tails and necks out on paper.

(2) Input the point for these objects.

(3) Build the tubular object around these input points.

(4) Translate the hips back to 0 (retaining the information for later placement).

(5) Create the timing sheet for a sequence.

(6) Write a twixt script that contained all the calls to changeshape and the position of the legs.

(7) Run the script through reattach.

(8) Create the appropriate path and run the animation script through pathwalker.

(9) Run the output script through a program that changes to the event driven twixt format.

(10) Finish animating the character; as head rotations, foot rotations, eyelid blinks and iris rotations.

(11) Feed Snoot's and Muttly's animation scripts together for final changes.

(12) Add the bubbles.

(13) Split the script back up into three separate scripts, one for Muttly, one for Snoot and one for the bubbles.

(14) Calculate the final frames.

(15) Overlay the frames on the electronic still storer.

(16) Record the sequences onto one inch video tape.

(17) Edit the tape together and overlay the sound.

A more complex leg cycle was created for Muttly, but never used. It was a turning dance sequence. The step went: right foot forward with a forty five degree turn to the left, left foot cross behind with a ninety degree turn to the left, step back left with a forty five degree turn to the left and then right leg around with at ninety degree turn. It was very difficult to set this action up without including the turn in the data. This is because of the constant change in step direction. If the data could have been described without the rotation inherent in it, the rotation would have been added in the animation script. The legs were built with the rotation inherent in the data. The problem was that the legs never actually changed position during the rotation. It appeared that the data was built with a rotation, however the character stepped left, right, right, left. This created a need to switch the position of the legs during the first 180 degrees of the turn. This was laid out on a map. It was discovered that the new attach points for all these legs would have to be defined by hand. The second problem was the number of legs it would take for the dance cycle. With

all the necessary key positions and inserts for the angle problem in changeshape there were 60 legs. The final problem with this animation was how to define a path to the pathwalker program so that the new positions and attaches would be transformed to stop the foot slippage. This sequence was eventually replaced with a simpler motion. This test pointed out the need for a new animation system for more complex character motions.

The characters in this animation do begin to have personalities, and there is a simple underlying story to the animation. The characters do however have very rigid faces and this needs to be changed before computer generated characters will have the illusion of living flesh. The animator needs to be able to spend a greater percentage of their time animating and a smaller percentage dealing with computer related problems. The solution to this problem includes the development of new hardware for real time display and a software package designed for character animation.

Figure 33.
Overlay of frames from Snoot and Muttly

There are two different ways of approaching a computer generated character animation system. The first is a low level system, the second is a high level system. A low level system would necessitate a great deal of animator input. A high level system would have more processes automated by the computer.

## CHAPTER 11

## A FUTURE LOW LEVEL CHARACTER ANIMATION SYSTEM

To create a character that appears to have living flesh, the parts of the character must be soft and pliable (plastic). A system to easily manipulate the motion of articulated figures is needed. There are a number of papers available on the manipulation of the hierarchical structures [19,10,4,11]. A system that could easily and more completely control the surfaces of plastic objects is also essential. There are also people working on surface manipulation however I know of no papers extant in the literature.

The low level system that I suggest is a two-stage system. The first stage is to animate of the skeleton of a character. The second stage is to animate the surface of that character. A simple skeleton representing all or most of the pieces of a character would be created. This would be quickly displayed on a vector display device creating the ability to see motion in near real time. This system would also need the capacity to handle more than one character at a time. This is especially important when the animator is trying to portray character interaction.

It is often difficult for the animator to tell if the character is facing directly toward the animator or directly away on a vector display device. This is because all the lines are displayed for the character. When lines that are defining unseen surfaces of an object are removed from the image it is much easier for the animator to tell what part of the character is closest to the viewer. The removal of these lines is called "hidden line removal" and is also desirable in a character animation system.

The event-driven format of twixt is a much better solution for setting up animation than the keyframe animation system. The ability to set one parameter as an event without effecting other parameters is extremely useful. The ability to do support and non-support rotations found in the skeleton animation system is also valuable. One of the problem which is common to both of these programs is

that rotations are expressed with relation to the Cartesian coordinate system. This geometric way of manipulating articulated characters can often become confusing. The order in which rotations are given can affect the final outcome of a character's position. For example, a rotation of the arm about the y axis could be a large swinging or a radial rotation of the characters arm. This depends on the rotations that have occurred previously [19]. A system has already been developed by the medical profession to describe the possible motions of human joints. There are only six types of rotation for any joint in the human body. These are flexion, extension, adduction, abduction, rotation and circumduction. If animators had access to these six commands they would animate with respect to the anatomy of the character, rather than with respect to the geometry of the coordinate system [15]. This is a much more direct way for the animator to interact with the computer. Once an animator has one step for a character completed, it is easy to repeat the step to form a cycle. This cycle becomes part of a library of motion by saving the information that describes that particular walk. The cycle could be used anywhere in any environment with the addition of a program similar to pathwalker. Without the ability to place characters along a path the animator must specify any variation in the character's step caused by the angle of the turn. The pathwalker program allows the character motion to be setup through the creation of only one step. The program takes care of all the variations that are caused by the curvature in the path. The animator simply inputs the curve that the character is to follow. The character is oriented forward along that path. Unfortunately a complication with this program occurs when a character has a secondary rotation along a path. For example, this would occur when a character is spinning during a dance sequence. The problem also exists if the character is to be stepping sideways, backwards or at an angle to the path of motion. This orientation problem could be solved, by allowing the animator to manipulate the rotation of the entire character. This rotation would supercede the orientation of the character as output by the pathwalker program for specified frames. Additionally the pathwalker type program should be available within the animation system, rather than as a post process.

Animating the surface of an object is a more difficult problem. Using the animation of Snoot and Muttly as a starting point, it is easy to see how a system to animate surfaces might evolve. The legs of the birds were created by accessing a program that ran a spline through three original input points. A program then built a tube around the splined points. To create animation the animator specified at what frame each 'key-leg' would be called. The rest of the process for the legs was controlled by running the animator's script through computer programs. These programs interpolated between key-legs, attached the feet to the legs and placed the birds correctly along paths. The location of the original three input points and the frame count were the only variables that created the locomotion of the birds. If these original input points were displayed with a line connecting them, they could be used as the skeleton of the leg. The basic motion could be set up by rotating or placing these points on an interactive vector display system. If the animation sequence of these three points was output directly to the leg building program, the actual leg surface could be generated during the final image rendering. A new leg would be generated for every frame. There would be no problems with the changeshape program, as it would not be used. This means that the original input point could be animated in three-dimensional space without complication. This also means that no legs would be stored on the system. Only the animation script would be stored.

The motion of the surface could be generated by manipulating the points of that surface, and/or the spline which travels through those points. The animator should have independent control of individual parts of the character's surfaces. The face might be the only surface manipulated on a character. The independent parts of the character would need to be attached with a smooth surface. This surface could be generated by running a spline across two connecting parts of the character. The spline may not behave in exactly the way an animator would like. Because of this the animator would need to have the ability to manipulate this spline.

It is necessary for the animator to have a simplified way of manipulating the points which define surfaces. The 'warp' program discussed in chapter three, could be used for manipulating surfaces. This type of manipulation may be hard to control in a manner exact enough for the

expression of emotions on a character's face. Instead, a set of tools could be created that, when intersected with an existing surface, would manipulate that surface. For example, pushing a sphere into the cheeks on the inside of a characters mouth would cause the cheeks of the character to be pushed up and out, forming the rounded cheeks of a smile. The types of tools necessary are pulling, pushing, creasing and knifing tools. The animator can build specially shaped tools, there would soon be a variety of tools for molding surfaces. If very subtle changes are needed the shape of the spline through these points could be manipulated. One example of this is outlined by D. Kochanek. [9] Although this paper outlines control of splines for generating motion control, the ideas could be transferred to surfaces generation as well. The manipulation of these points for every object could become extremely time consuming and confusing. To simplify the process the animator could have a library of pre-defined surfaces. For example, there might be two definitions for the surface of the arm. One surface describes the arm when it is straight and relaxed, the other when it is flexed showing a large bicep. The computer could interpolate between these two sets of points to locate the proper surface associated with the current rotation at the elbow. The flexing of the surface of the arm might be used to animate a character like Popeye.

An animation system that could associate surfaces of a character by the degree of rotation about specific joints would need to have a knowledge of the structure of the character. This knowledge would be incorporated in a higher level animation system.

# CHAPTER 12

## HIGHER LEVEL CHARACTER ANIMATION SYSTEMS

A high-level character animation system refers to a system that has knowledge of the structure of the elements in the animation environment. When an animator must specify all of the motion and surface descriptions for an animation, they are working at a lower level than is possible. The degree of knowledge within a system can vary greatly. In a high-level system the animator can deal with a parameterized model of an object rather than with the objects themselves. A simple example of this is the bubbles in Snoot and Muttly. The bubbles were placed within a four by four by seven grid by a computer program. The animator gave the program one path along which the nine bubbles moved. The program placed the bubbles at the vertexes of the grid which traveled along the specified path. A parameter was used to control the frequency at which the bubbles moved from vertex to vertex within the grid. The legs of the bird have a similar property. The walk was a cycle created from one original step. A computer program placed the legs along a specified path. Programs such as these allow the animators to concern themselves with other aspects of the animation.

A high-level system should be designed with a modular structure. Animators working on the lower levels of this modular system may find that they repeatedly use a particular motion. This motion would be incorporated as a parameter of the higher levels of this system. In this way a high-level system evolves from the needs of the animators.

There is a great number and variety of parameterized models that can be created. Animators must have the ability to create their own models if they are to work independently from the computer scientist on a day to day basis. A visual interface would be the most natural for animators. One possibility is a visual data flow environment similar to the PIGGS programming language [14].

If a set of default parameters exists within a character animation system, the animator needs the freedom to manipulate these parameters to create an individual

character. In some cases it is more work to over ride automatic high-level systems than it is to start at the low level system and work up. This would depend on the length of an animation and the number of parameters that must be manipulated to take a default motion and change it into one that has the necessary motion of a individual character.

The skeleton animation language is a high-level system. The animator can tell a character to walk and the character will walk correctly whether the ground is level or uneven. This is done by accessing a predefined walk sequence. The model is programmed to detect when the foot touches the ground. The model could also be used to have the character avoid obstacles along its path. Once a library of motions is created the character is animated by specifying goals for the character to reach. A goal directed command is: walk to the refrigerator and grab a beer. The program would calculate the location of obstacles and manipulate the character to avoid them [19]. There are different ways of controlling how the character grasps the beer. One way is to call a predefined library for a definition of "grasp". Another implementation would be the application of inverse kinematics. Inverse kinematics are used in robotics to manipulate the arms of a robots. In this case: the animator specifies where the beer is and what the preferred angles are for each joint. The computer calculates the dynamic joint rotation which achieves the desired goal.

Another working parameterized model for characters exists at the robotics lab at The Ohio State University. The character is capable of walking up a specific set of stairs. The character's motions are changed by specifying the new depth and rise of the stairs [12].

Properties of the physical world have also been modeled. For example, gravitational force can be a parameter within an animation system. The effect of gravity on an object can be calculated by the computer system. This means that the velocity of a bouncing ball would be calculated by the computer. The resilience of the ball and the floor would be a parameter within this system. Another parameter would be used to specify the amount of squash and stretch the ball displays. The computer could mathematically determine the angle at which the ball would leave an uneven surface.

A research project at C.G.R.G. deals with changing the gait of characters by changing a parameter that represents the amount of force a character uses to push off the ground. The distance that the character travels is changed by manipulating this variable rather than by placing the character [6].

Obstacle avoidance amongst objects can also be achieved through the use of force fields. An animation is underway at C.G.R.G. that uses a program to control the motion of birds. The birds are moving toward an attracter a parameter is used to define force fields that repels the objects from each other as they move through the environment [1].

In a similar manner other physical models could be made. The number of rotations which wheels of an automobile make to travel a specified distance is directly related to the circumference of that wheel. An animator specifies a path for a vehicle to travel and the computer calculates the number of rotations the wheel would make to travel the distance of that path [10].

Motion for an animation could be described to the computer with written motion languages. Dance choreographers use a number of different languages to document dances. Input from these languages could be used to define the motion of a character. Motion can also be taken directly from a person performing the desired motion. There are different ways this motion can be recorded and used as input to computer character animation systems. The Ohio State University Robotics Lab used light emitting diodes attached to a person. The diodes were tracked with two cameras set up at right angles to each other. In these ways the motion of a person is input to the computer in three dimensional space. This input motion can be associated with a specific character. The motion can be manipulated to create deviations from the original motion for specific characters.

These types of parameterized models can also be used to generate the surface of a character. Instead of using tools described in the previous chapter, surfaces could be manipulated with parameterized models associated with those surfaces. One of the desired parameters for manipulating surfaces is a parameter that maintains the volume of an object. When the length of a characters limb is scaled, the program maintains the volume of the object .

BEST COPY AVAILABLE

By manipulating the parameters that maintain volume, accurate squash and stretch is achievable.

In a high level system the animator is not working directly with the character. Instead they work with mathematical models that control parameters on a character. It is possible that an animator would have less creative input with a high-level system. However a well designed system would not necessarily have this drawback. The animator needs interactive access at both the high and low-levels. The high-level system helps the animator by looking after many of the tedious details of a complex animation. As long as animators have the capability of controlling the results of parameterized models, they are not loosing creative control of animation.

# CHAPTER 13

## CONCLUSION

The quality of conventional animation often suffers because the technique required is laborious and time-consuming. The computer has the ability to reduce this tedium, thus allowing the art of animation to grow. It is generally assumed that the cost of computer hardware will continue to decrease. Additionally as friendly software becomes available the computer will become a tool that helps animation evolve rather than a technical threat to the art of animation.

Computer animation is not a separate entity, but a new capability that exists within the film and video industries. Today many of the computer generated characters are demonstrations of technical accomplishments. As this technology becomes more readily available, it will be used as a tool to communicate information and ideas. When computer graphics characters are used for communicating they will need to live, think and feel. The characters will have personalities that will be portrayed by both their gestures and their posture. The computer animator will need to have a good understanding of principles found in the performing arts, conventional animation and cinema. Those who concentrate their studies in these artistic endeavors often can not also find time to become proficient as mathematicians or computer scientists. Because of this animators and programmers must work together trying to understand each others needs and limitations. For the animator this means understanding the current and potential advantages and disadvantages of computer animation. These advantages and disadvantages will change as software is written that addresses problems specific to character animation. The two major problems in this area are the control of hierarchical objects and the manipulation of plastic surfaces.

Character animation languages will revolutionize the speed at which character animation can be produced. In order for new animation languages to improve the quality of animation that can be produced the animator will need

to be able to do five things:

(1) Build the original character.

(2) Specify its hierarchy.

(3) Define the original motion.

(4) Easily change the motion.

(5) Create individualized parameterized models.

The animator also needs to be able to work with the surface of a character in these ways:

(1) Build the original surface description.

(2) Manipulate the surface of individual character parts.

(3) Manipulate the surface as a whole.

(4) View the surface when needed during the animation process.

A computer system that has the capabilities of manipulating hierarchical objects and surface definitions is not a character animation system. It is a system for the manipulation of objects. The talent to manipulate these objects in a manner which creates personality and life must accompany it. The people who have this talent must be able to interface with this system at all levels. Only when the people with the talent can control the computer system, does a three dimensional character animation system exist.

## List of References


1. Amkraut, Susan, 1984. personal communication
2. Blinn, James F., "Systems Aspects of Computer Image Synthesis and Computer Animation," in Three-Dimensional Computer Animation Tutorial Notes, SIGGRAPH-ACM (1982).
3. Carlson, Wayne, "Techniques for the Generation of Three Dimensional Data For Use in Complex Image synthesis," unpublished Ph.D. Dissertation , The Ohio State University (September, 1982).
4. Crow, Franklin, "Scn_assmblr," in UNIX Programmer's Manual, Local Addendum, Computer Graphics Research Group, The Ohio State University , Columbus, Ohio (). in progress
5. Fortin, Denis, Lamy, Jean-Francois, and Thalman, Daniel, "A Multiple Track Animator System for Motion Synchronization". Proc. ACM SIGGRAPH/SIGART Workshop on Motion, (April, 1983), pp. 180-186.
6. Girard, Michael, 1984. personal communication
7. Gomez, Julian, Keyframe Twixt. Computer Graphics Research Group, The Ohio State University, Columbus, Ohio (1983). An undocumented animation system based on keyframes.
8. Gomez, Julian, Computer Display of Time Variant Functions. The Ohio State University, Columbus, Ohio (1985). unpublished Ph.D. Dissertation; in progress.
9. Kochanek, Doris H. and Bartels, Richard H., "Interpolating Splines with Local Tension, Continuity, and Bias Control". Computer Graphics, 18, (3) (July, 1984)Proc. ACM SIGGRAPH 84.
10. Lundin, Richard V., "Motion Simulation". Nicograph 1984 Proceedings, (November, 1984), pp. 2-10.

11. Reynolds, Craig, "Computer Animation with Scripts and Actors". Computer Graphics, 16, (3) (July, 1982), pp. 289-296. Proc. ACM SIGGRAPH 81.

12. Ribble, E. A., "Synthesis of Human Skeletal Motion and the Design of a Special-Purpose Processor for Real-Time Animation of Human and Animal Figure Motion," M.S. Thesis , The Ohio State University (June, 1982).

13. Schweppe, Marla, "Timing and Movement in 3-D Computer Animation," unpublished M.A. Thesis , The Ohio State University (March, 1984).

14. Sklar, Dave, PIGGS, The Polyiconic Interactive Graphical Generation System. Brown University, (1983). unpublished.

15. Stredney, Donald, "The Representation of Anatomical Structures Through Computer Animation for Scientific, Educational and Artistic Applications.," unpublished M.A. Thesis , The Ohio State University (March, 1982).

16. Thomas, Frank and Johnston, Ollie, Disney Animation: The Illusion of Life. Abbeville Press, New York (1981).

17. Van Baerle, Susan, The Uneven Bars. The Computer Graphics Research Group, The Ohio State University, (1983). SIGGRAPH Video Review 1983.

18. Van Baerle, Susan and Kingsbury, Doug, Snoot and Muttly. The Computer Graphics Research Group, The Ohio State University, (1984). SIGGRAPH Video Review 1984.

19. Zeltzer, David, Representation and Control of Three Dimensional Computer Animated Figures. The Ohio State University, (August, 1984). unpublished Ph.D. Dissertation.

## BIBLIOGRAPHY


Blair, Preston, Animation, Walter Foster Art Books, Tustin, CA.

Brown, Maxine, and Smoliar, S., "A Graphics Editor for Labanotation," Computer Graphics, 10, (2) (1976), pp.60-65. Proc. ACM SIGGRAPH 76.

Badler, Norman I., "Design of a Human Movement Representation Incorporating Dynamics," Course Notes, Seminar on Three-Dimensional Computer Animation ACM SIGGRAPH 82, Boston, MA. (July, 1982).

Chuang, R. and Entis, G. "3-D Shaded Computer Animation -- Step-by-Step," IEEE Computer Graphics and Applications, 3, (9) (December, 1983) pp. 18-25.

Csuri, Charles, et al. "Towards an Interactive High Visual Complexity Animation System," Computer Graphics, 13, (2) (1979).

Csuri, Charles, Gomez, Julian, MacDougal, Paul, Zeltzer, David, "A Tool Set for 3-D Computer Animation," Course Notes, Introduction to Computer Animation, ACM SIGGRAPH 84, Minneapolis, MN. (July, 1984).

Catmull, Edwin, "The Problems of Computer-Assisted Animation," Computer Graphics 12, (3) (July, 1978) Proc. ACM SIGGRAPH 78.

Crow, Franklin, "Shaded Computer Graphics in the Entertainment Industry," Computer 11, (3) (March 1978) pp.11-22.

Ginsberg, C. and Maxwell, D., "Graphical Marionette," Proc. ACM SIGGRAPH/SIGART Workshop on Motion, Toronto, Canada (April, 1983) pp. 172-179.

Gomez, Julian, "shapex" UNIX Programmer's Manual, Local Addendum, Computer Graphics Research Group, The Ohio State University, Columbus, Ohio, (October, 1983).

Gomez, Julian, "Twixt: A 3D Animation System," Proc. Eurographics '84 Copenhagen, (September, 1984).

Gomez, Julian, Twixt Reference Manual, Computer Graphics Research Group, The Ohio State University, (1984) (in progress).

Halas, John, Computer Animation, Visual Communications Book, New York, (1974).

Halas, John and Manvell, Roger, The Technique of Film Animation, Hastings House Publishers, New York, (1959).

Maciejewski, T. and Klein, C., Motion Control of Articulated Objects, Ohio State University Dept. of Electrical Engineering, Columbus, Ohio, (1984) (Unpublished).

Masden, Roy, Animated Film: Concepts, Methods, Uses, Interland Publishing Inc., New York, (1969).

Muybridge, Eadweard, The Human Figure in Motion, Dover, New York, (1955).

Muybridge, Eadweard, Animals in Motion, Dover, New York, (1957).

Parke, Fred, "Parameterized Models for Facial Animation," IEEE Computer Graphics and Applications, 2, (9)(November, 1982).

Parke, Fred, A Parametric Model for Human Faces, Ph.D. Thesis, University of Utah, Salt Lake City, (1975).

Platt, Stephen M. and Badler, Norman I., "Animating Facial Expressions," Computer Graphics 15 (3) (August 1981). Proc. ACM SIGGRAPH 81

Sachter, Judy, The Basic Concepts of Three-Dimensional Computer Graphics for Artists, unpublished Masters Thesis, The Ohio State University, (June 1984)

Whitaker, H. Halas, J., Timing for Animation, Focal Press Limited, London, (1981).

Williams, Lance, "BBOP", Course Notes, Seminar on Three-Dimensional Computer Animation, "ACM SIGGRAPH 82", Boston, MA. (July, 1982).

Zeltzer, David and Charles Csuri, "Goal-Directed Movement Simulation," Proc. Conf. Canadian Society for Man-Machine Interaction, (June, 1981).

Zeltzer, David, "Representation of Complex Animated Figures," Proc. Graphics Interface 82, Toronto, Canada. (May, 1982) pp.187-192,

Zeltzer, David, Three Dimensional Movement Simulation, "Course Notes, Seminar on Three Dimensional Computer Animation", "ACM SIGGRAPH 82", Boston Ma., (July, 1982).

Zeltzer, David, "Motor Control Techniques for Figure Animation," IEEE Computer Graphics and Applications, 2, (9) (November, 1982).

Zeltzer, David, "Issues in 3-D Computer Character Animation," "Course Notes, Introduction to Computer Animation", ACM SIGGRAPH 84, Minneapolis, MN. (July, 1984).